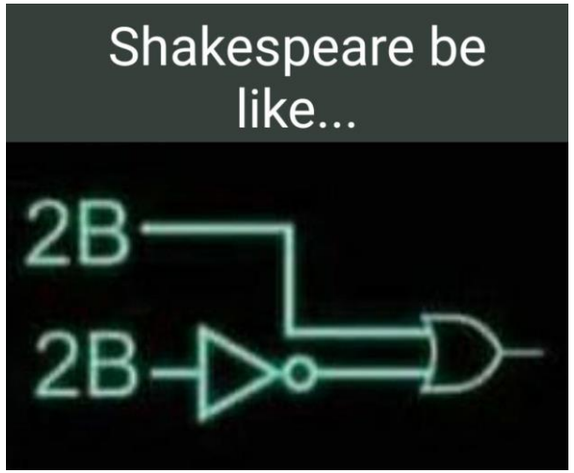


EECS 370 - Lecture 9

Sequential Logic

Poll: What is NOR(0,X)?

Announcements

- P2
  - P2a due next Thursday
  - P2L due in a month

- HW 2
  - Posted, due in ~2 weeks
- Lab 4 due Wed
- My group office hours moved to **4941 BBB** for this week only
- Midterm Exam
  - Thu Oct 12th, 6-8 pm (little over 2 weeks from today)

Next few lectures: Digital Logic

- Lectures 1-7:
  - LC2K and ARMv8/LEGv8 ISAs
  - Converting C to Assembly
  - Function Calls
  - Linking

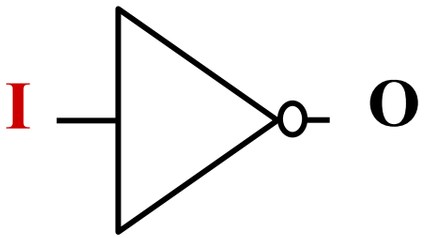
- Lecture 8:
  - Finish up linking
  - Combinational Logic

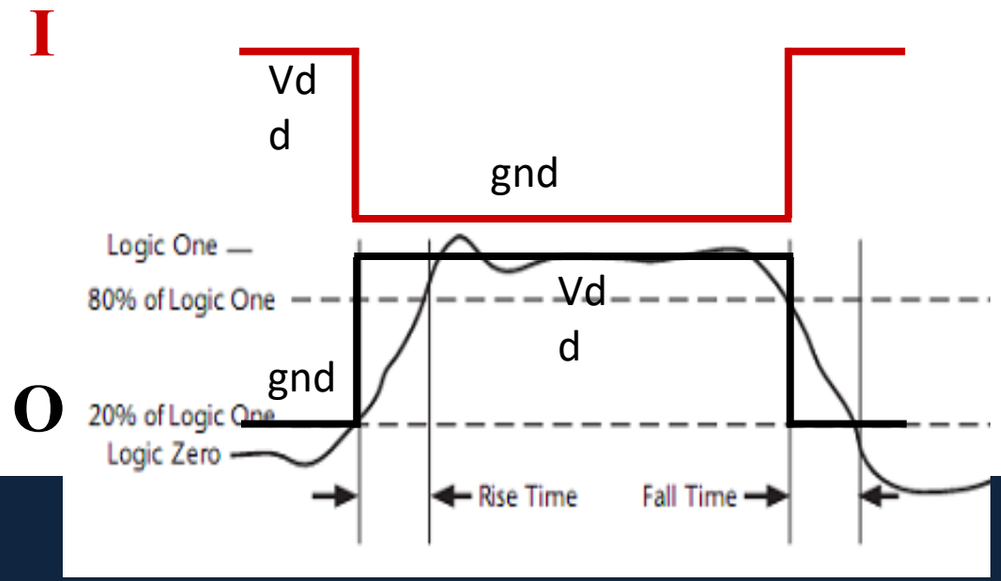
- **Today:**
  - **Sequential Logic**

Propagation delay in combinational gates

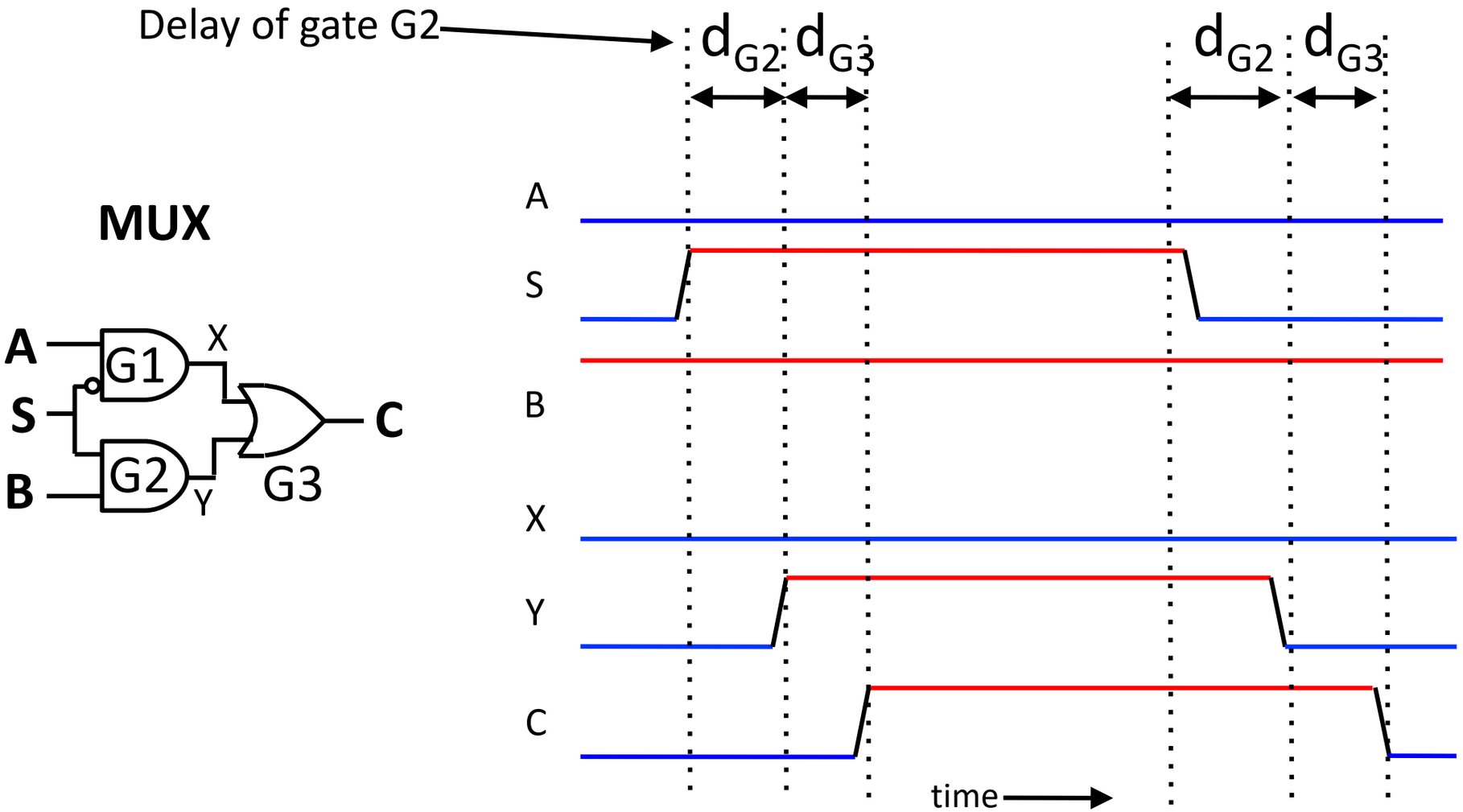
- Gate outputs do not change exactly when inputs do.
  - Transmission time over wires (~speed of light)
  - Saturation time to make transistor gate switch

  ⇒ Every combinatorial circuit has a propagation delay (time between input and output stabilization)





Timing in Combinational Circuits

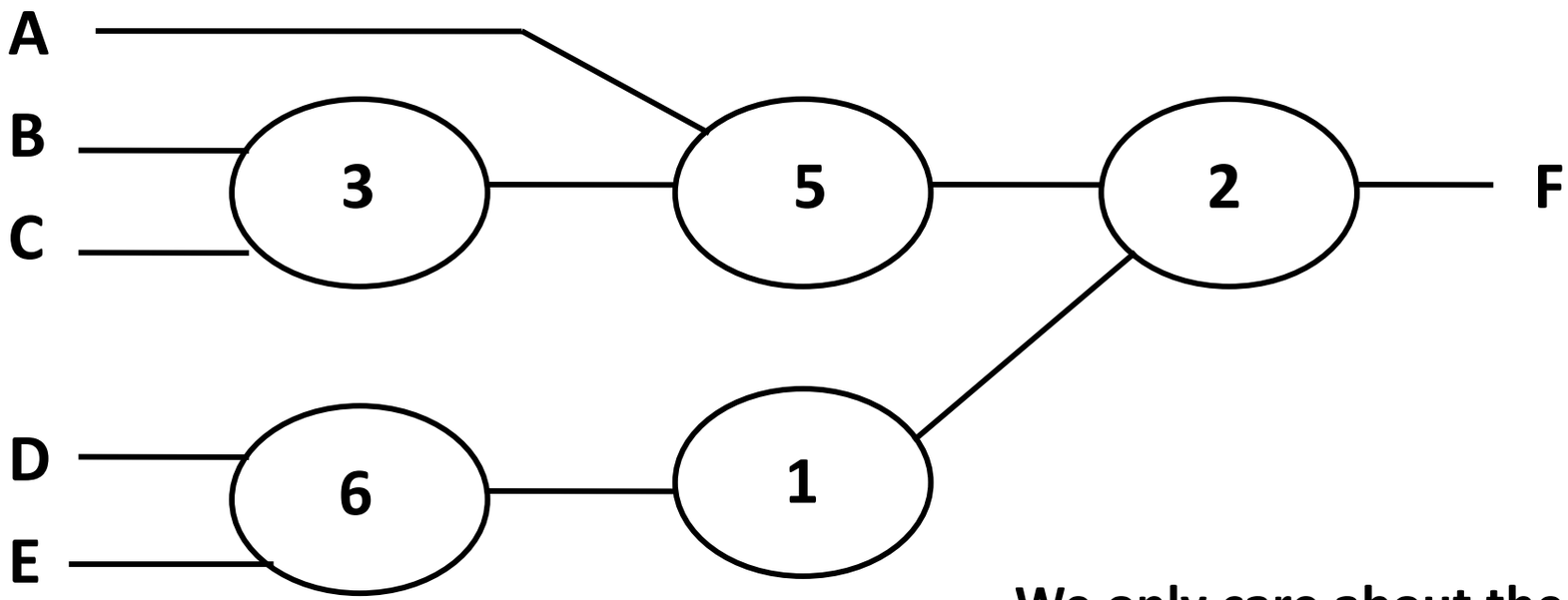


What is the input/output delay (or simply, delay) of the MUX?

What is the delay of this Circuit?

Each oval represents one gate, the type does not matter

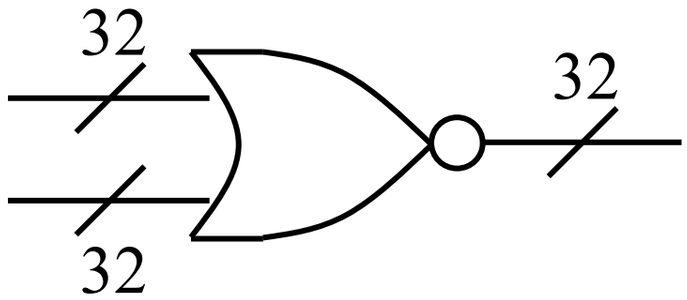
Poll : What is the delay?

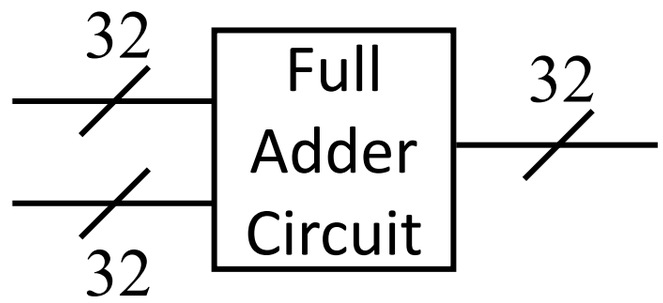


We only care about the longest path, or critical path

Exercise

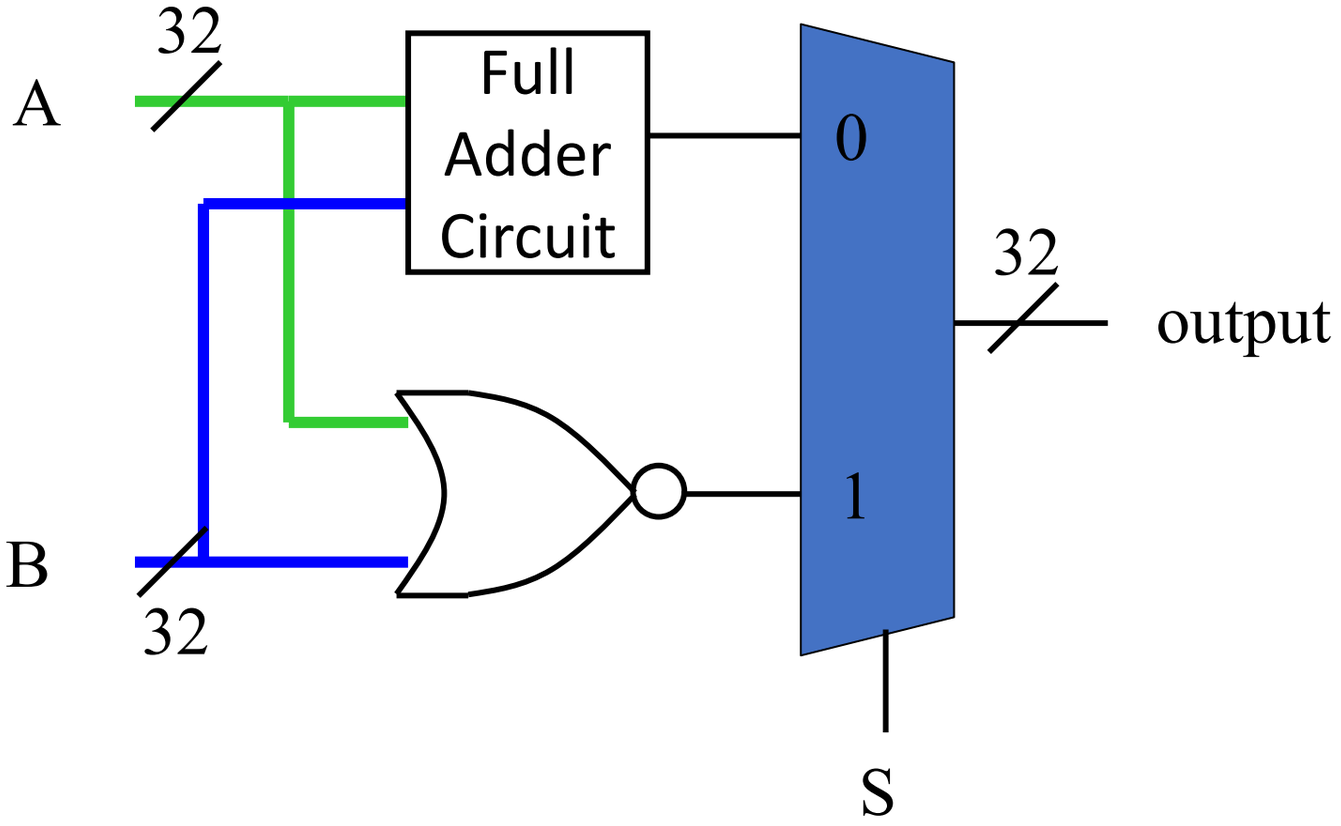
- Use the blocks we have learned about so far (full adder, NOR, mux) to build this circuit
  - Input A, 32 bits
  - Input B, 32 bits
  - Input S, 1 bit
  - Output, 32 bits
  - When S is low, the output is A+B, when S is high, the output is NOR(a,b)
- Hint: you can express multi-bit gates like this:



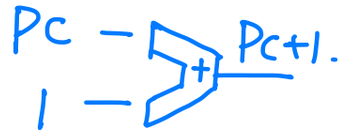


Exercise

- This is a basic ALU (Arithmetic Logic Unit)
- It is the heart of a computer processor!



Sequential Logic



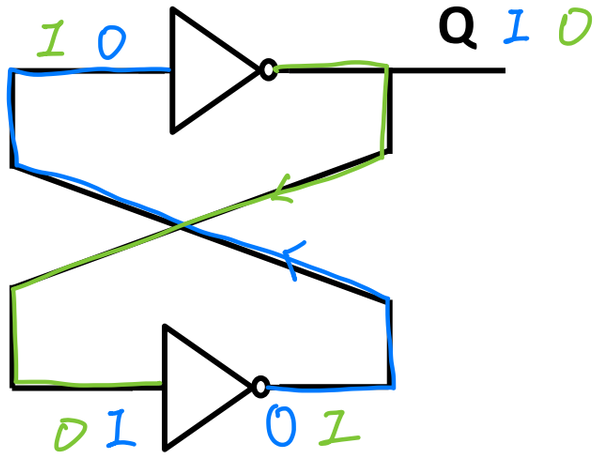
But we need to know what we do before.
which means we need to memory the state.

- Can we build a processor out of these combinational elements?
- How to build something like a program counter (PC)?
  - Increment it for every instruction… fine, use an adder
  - But only increment it once ready to move on to next instruction
  - That takes a finite amount of time… until then we need to "remember" the current value
- Combinational logic's output is determined from current input
  - But computers have "state" – they remember previous inputs and behave differently based on its history

Sequential Logic

- Examples of state
  - Registers
  - Memory
  - PC
- Sequential logic's output depends not only on current input, but also the current state
- This lecture will show you how to build sequential logic from gates
  - Key is feedback

Using feedback to "remember"



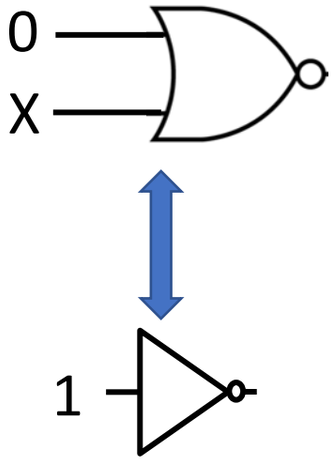
- This remembers its initial value!
- Very basic memory
- What's wrong with this, though?

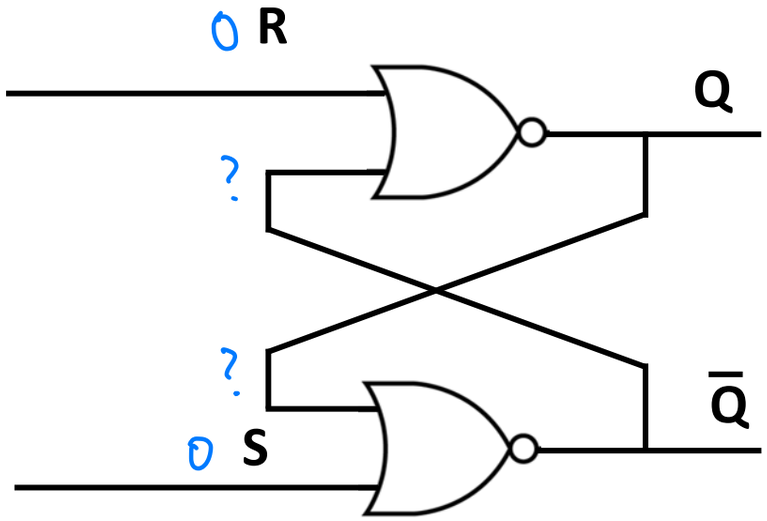
Not very useful for us. Since there is no way to control what the Q will be.

initial: 0 ⟶ Q: 1
initial: 1 ⟶ Q: 0

Let's look at the following circuit

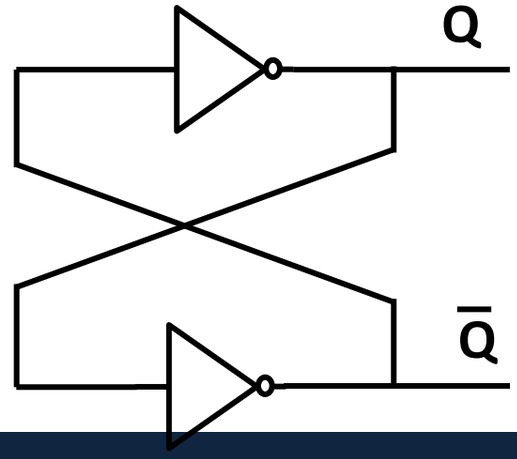
Useful identify:
NOR(0,X)=
NOT(X)





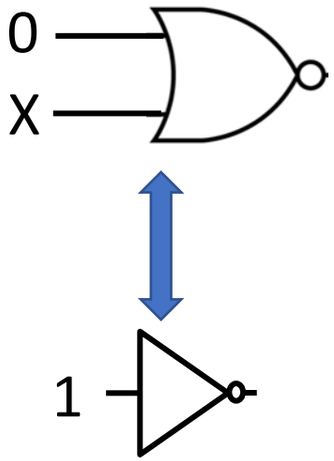
S	R	Q	Q̄
0	0	Q	Q̄
0	1		
1	0		
1	1		

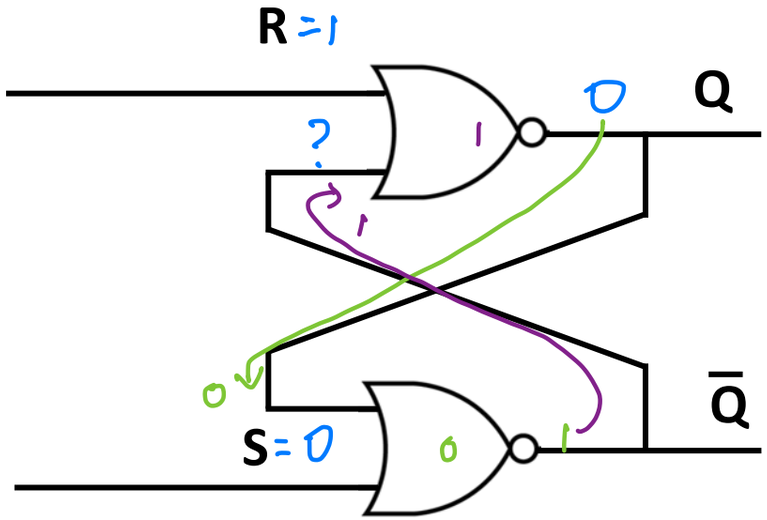
When R=S=0, turns into our memory element!



Let's look at the following circuit

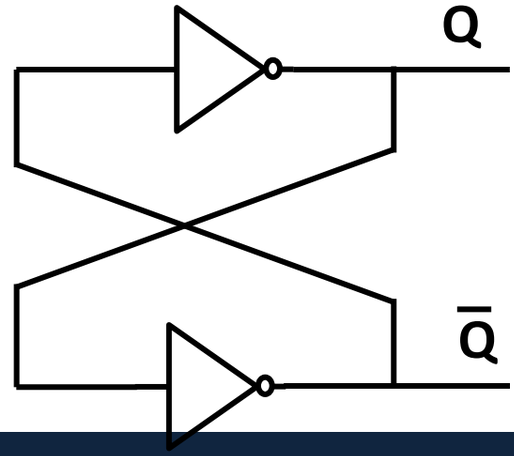
Useful identify:
NOR(0,X)=
NOT(X)





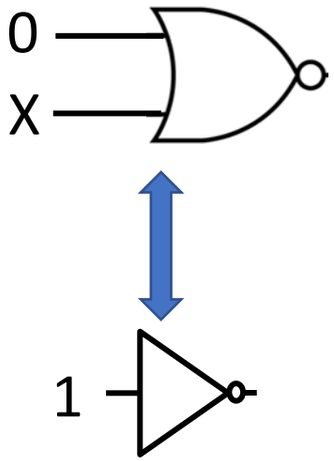
S	R	Q	Q̄
0	0	Q	Q̄
(stable) 0	1	0	1

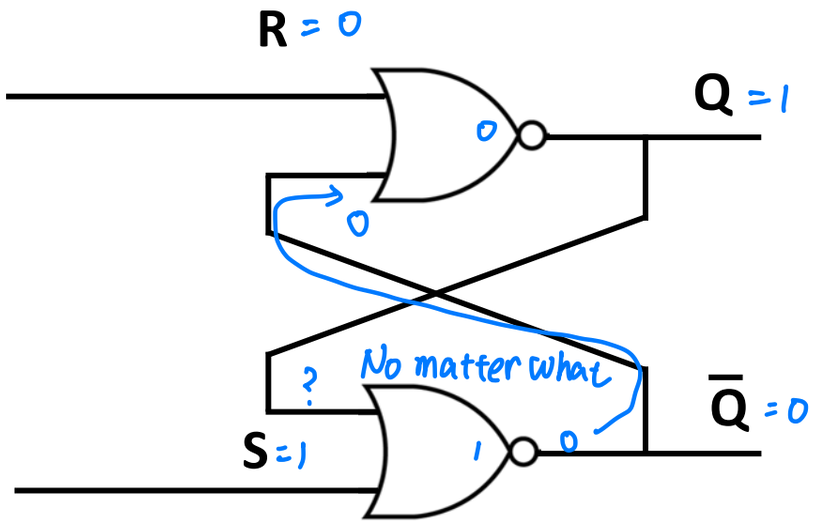
When R=S=0, turns into our memory element!



Let's look at the following circuit

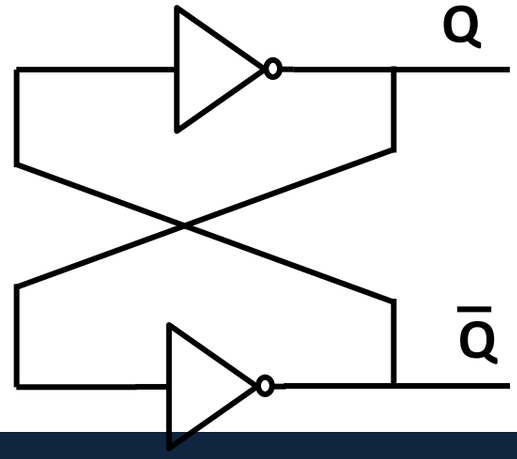
Useful identify:
NOR(0,X)=
NOT(X)





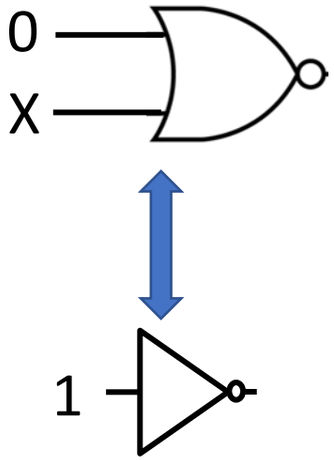
S	R	Q	Q̄
0	0	Q	Q̄
(stable) 1	0	1	0

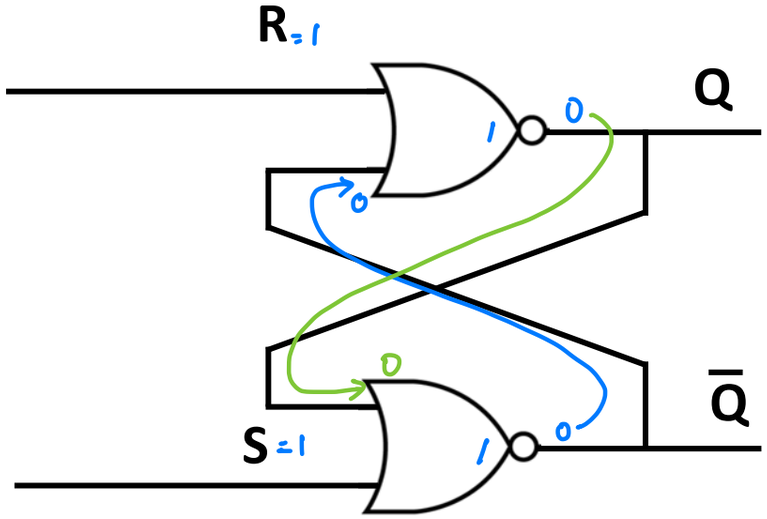
When R=S=0, turns into our memory element!



Let's look at the following circuit

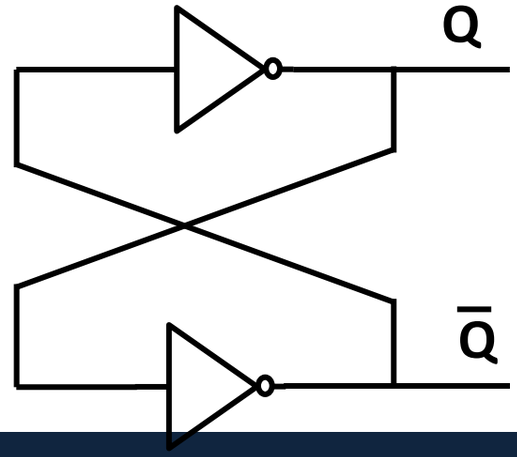
Useful identify:
NOR(0,X)=
NOT(X)



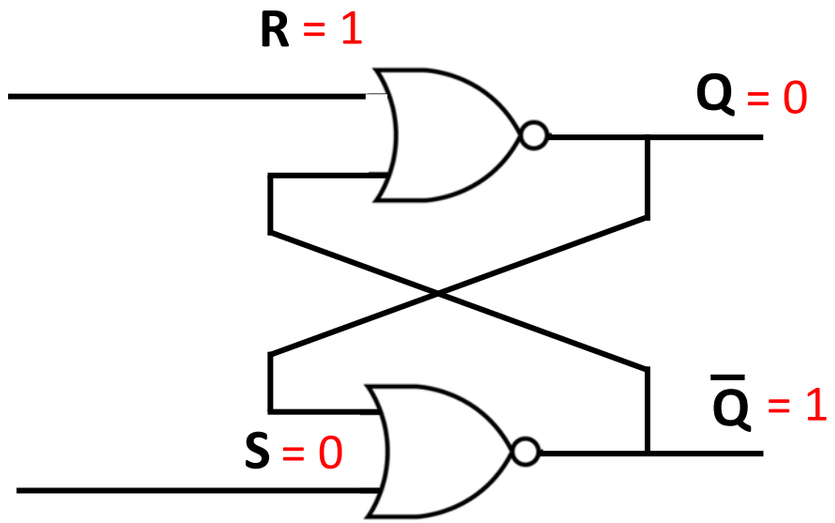


S	R	Q	Q̄
0	0	Q	Q̄
1	1	0	0

When R=S=0, turns into our memory element!



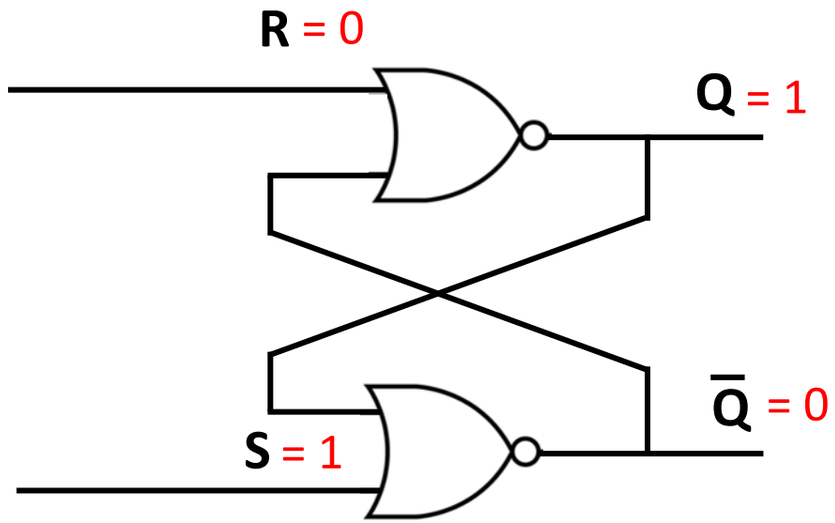
Let's look at the following circuit



S	R	Q	$\bar{Q}$
0	0	Q	Q
0	1	0	1

What is the value of Q if R is 1 and S is 0?

Let's look at the following circuit

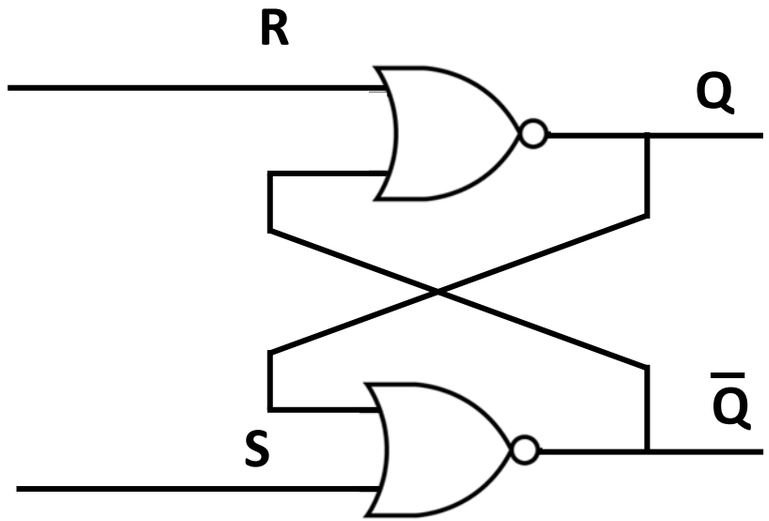


S	R	Q	$\bar{Q}$
0	0	Q	Q
0	1	0	1
1	0	1	0

What is the value of Q if R is 0 and S is 1?

SR Latch

- So this circuit (an SR latch):
  - "Sets" Q to 1 when S=1 R=0
  - "Resets" Q to 0 when S=0 R=1
  - "Latches" Q when S=0 R=0
  - What about when S=1 R=1 ?



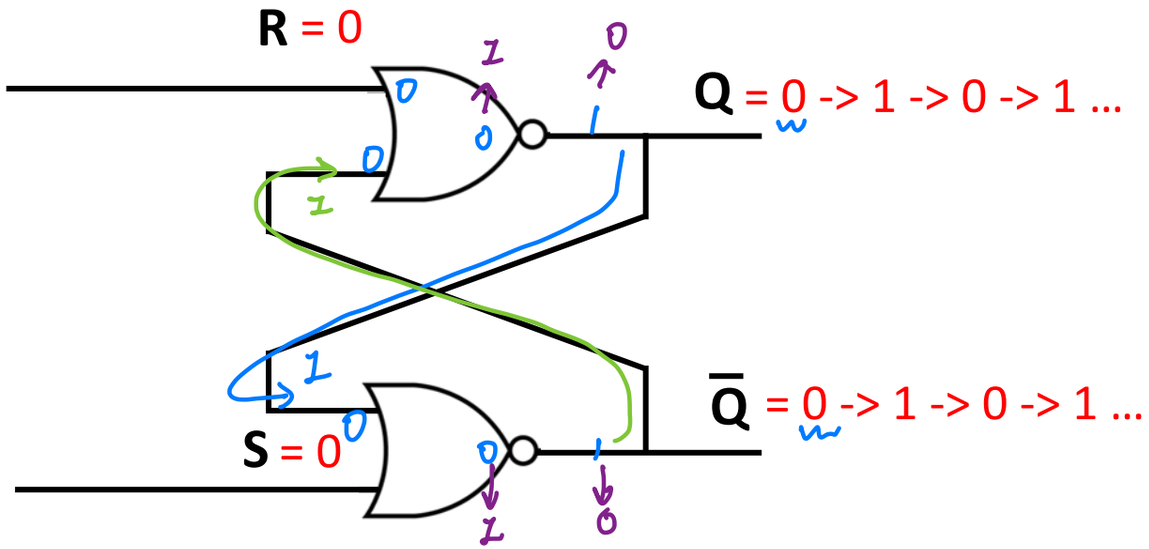
S	R	Q	$\bar{Q}$
0	0	Q	$\bar{Q}$
0	1	0	1
1	0	1	0
1	1	0	0

it's stable But Q and $\bar{Q}$ should inverse

BAD! Why?

SR Latch – Undefined behavior

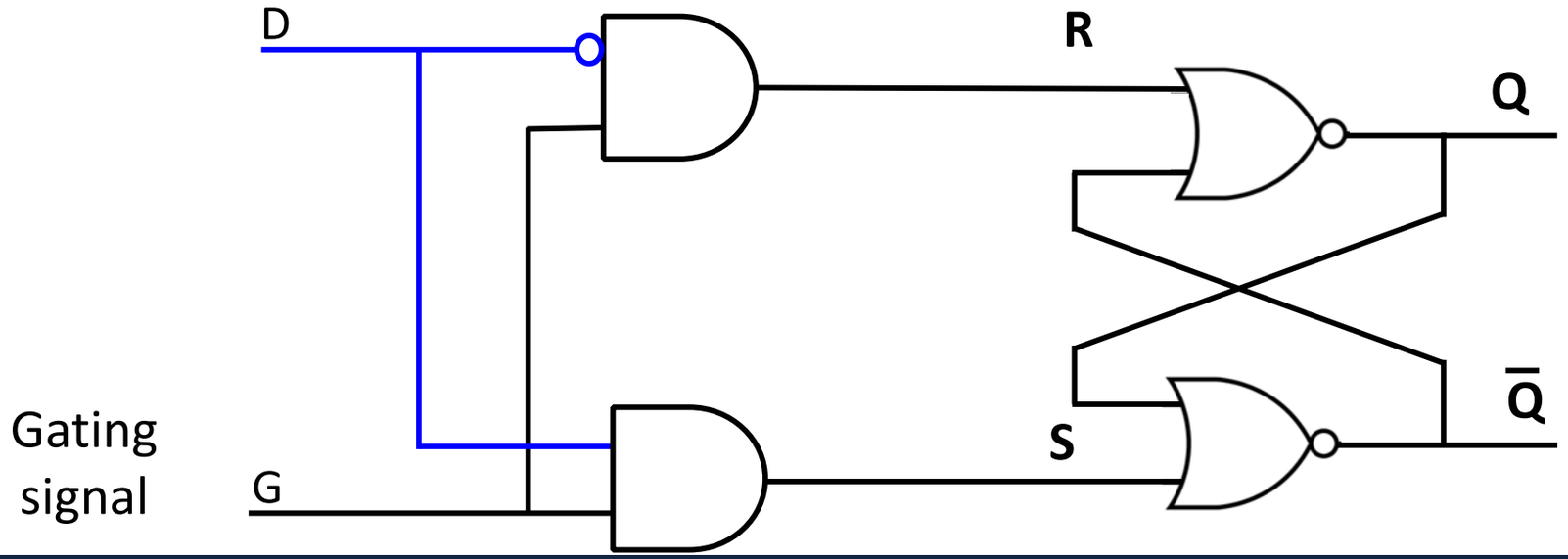
- If S=1, R=1, then Q and it's inverse are both 0
- If inputs then change to S=0, R=0, we get this circuit



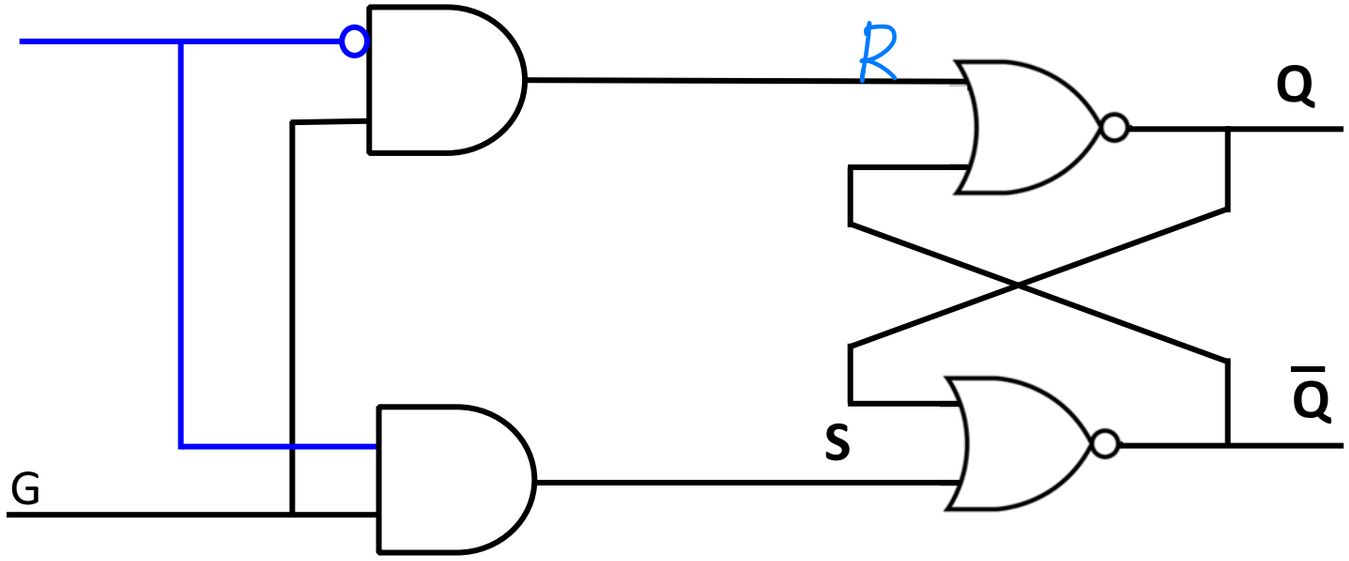
- This is unstable! Output rapidly oscillates between 0 and 1

Improving SR Latch

- SR Latch works great at saving a bit of data…
  - Unless S=R=1, even for a fraction of a second
- Idea: let's prevent that from happening by adding AND gates in front of each input
  - Impossible for R and S to both be 1



Transparent D Latch



Gating signal

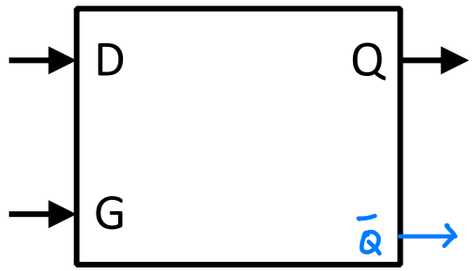
when G=0, it's like we close the Gate.

D	G	Q	$\bar{Q}$
0	0	Q	$\bar{Q}$
0	1	0	1
1	0	Q	$\bar{Q}$
1	1	1	0

Next state is set

Set state is retained when gate is low

if G=1, then Q is equal to D.



No invalid states ☺

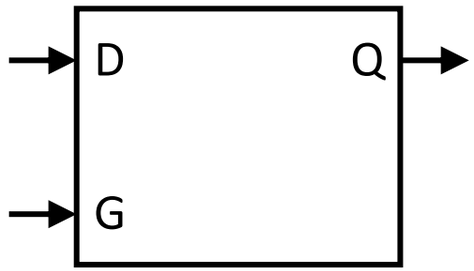
Transparent D Latch

- When G ("gate") is high, Q=D (the latch is "transparent")
- When G is low, Q "latches" to the value of D at that instant and remembers, even if D changes later

D	G	Q	$\bar{Q}$
0	0	Q	$\bar{Q}$
0	1	0	1
1	0	Q	$\bar{Q}$
1	1	1	0

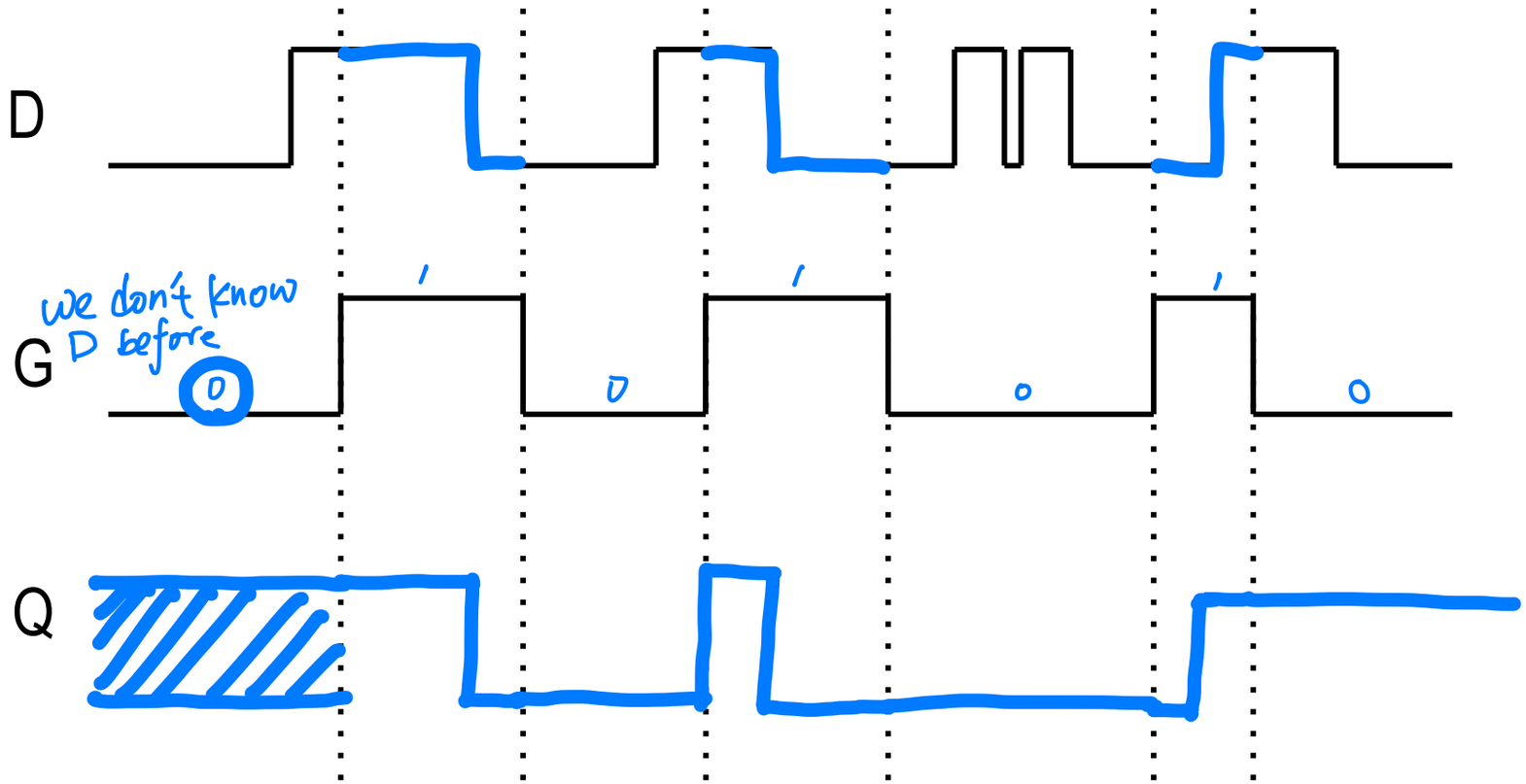
Next state is set

Set state is retained when gate is low

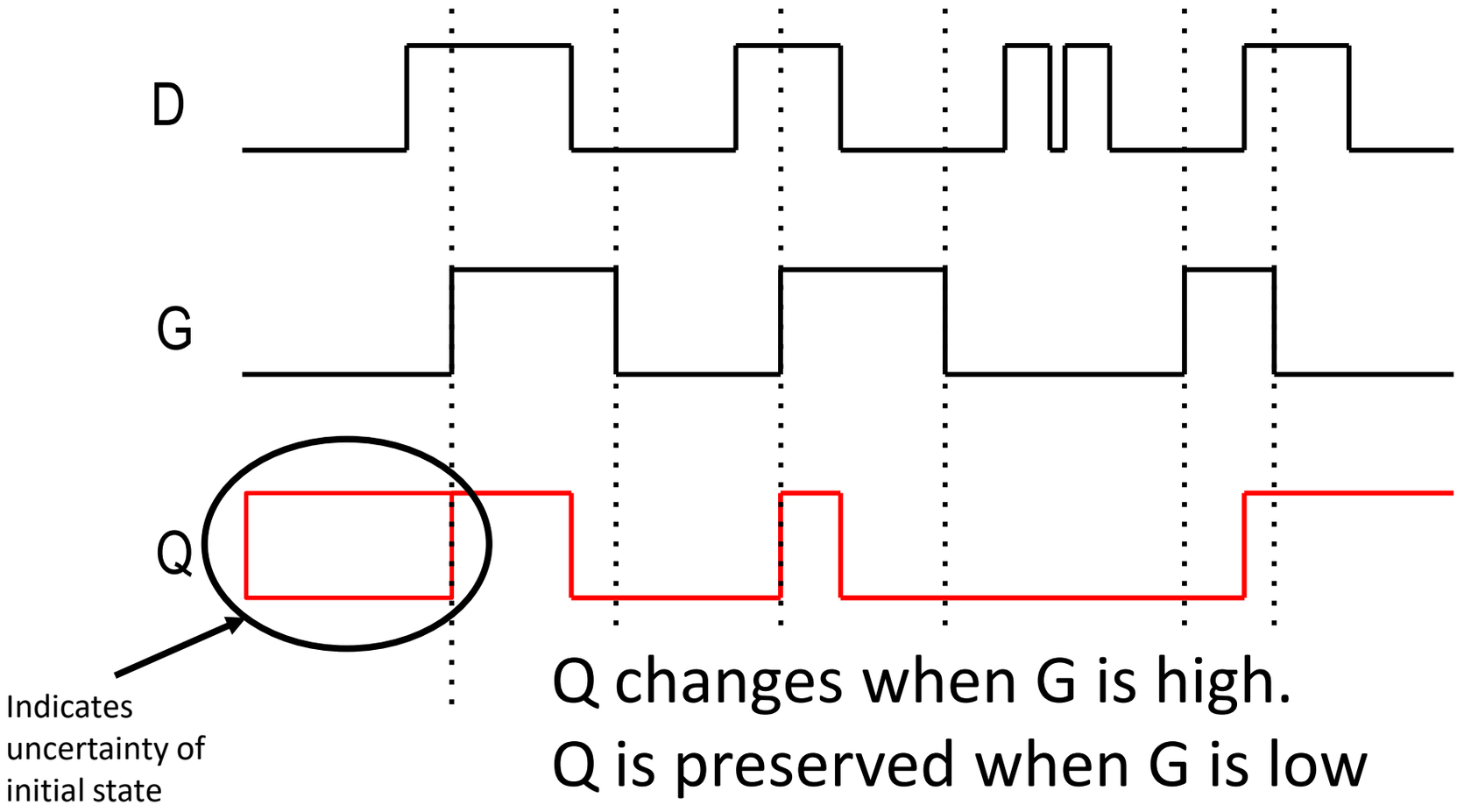


No invalid states ☺

D-Latch Timing Diagram



D-Latch Timing Diagram



Is D-Latch Sufficient?

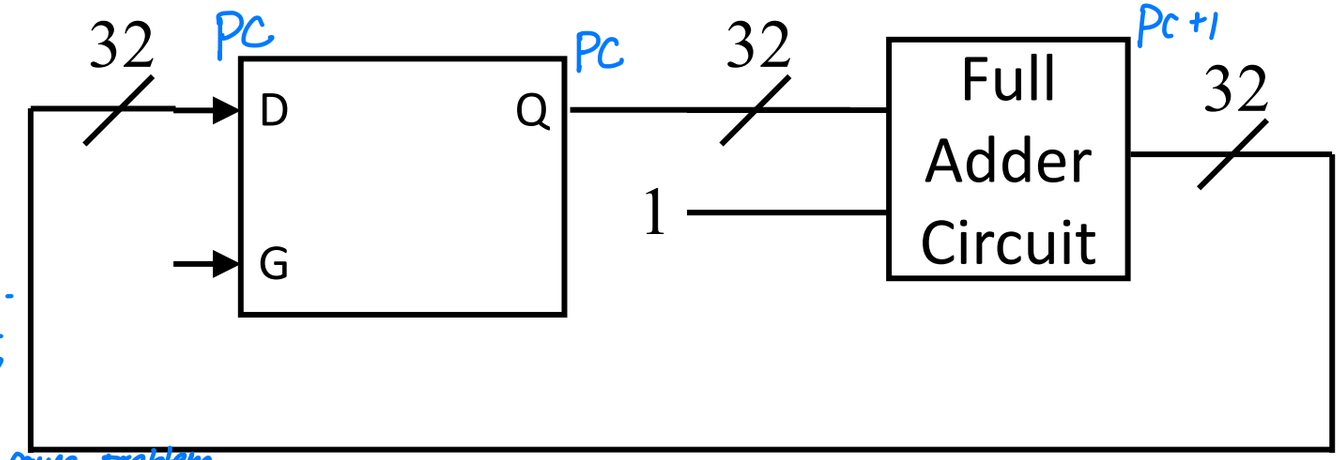
- Can we use D-latches to build our PC logic?
- Idea:
  - Use 32 latches to hold current PC, send output Q to memory
  - Also pass output Q into 32-bit adder to increment by 1 (for word-addressable system)
  - Wrap sum around back into D as "next PC"
  - Once ready to execute next instruction, set G high to update

we want G: ⎍

But the high frequency is really hard.

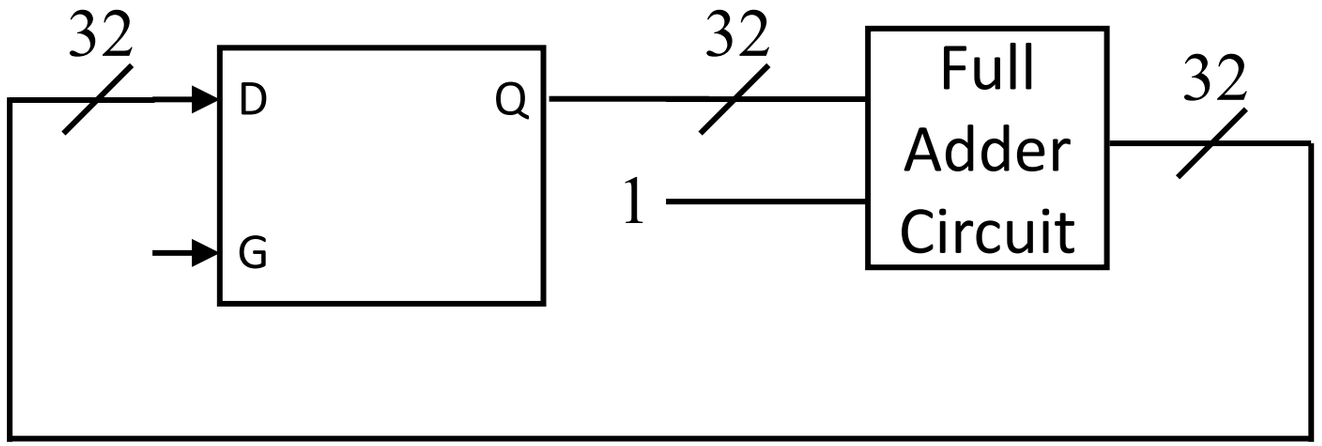
Also, if G is in high voltage for longer time, then the loop may run more time than we expected.

Also it will take constant time to Go through the D-latch So if G in high voltage too short, it will also cause problem.



Shortcoming of D-Latch

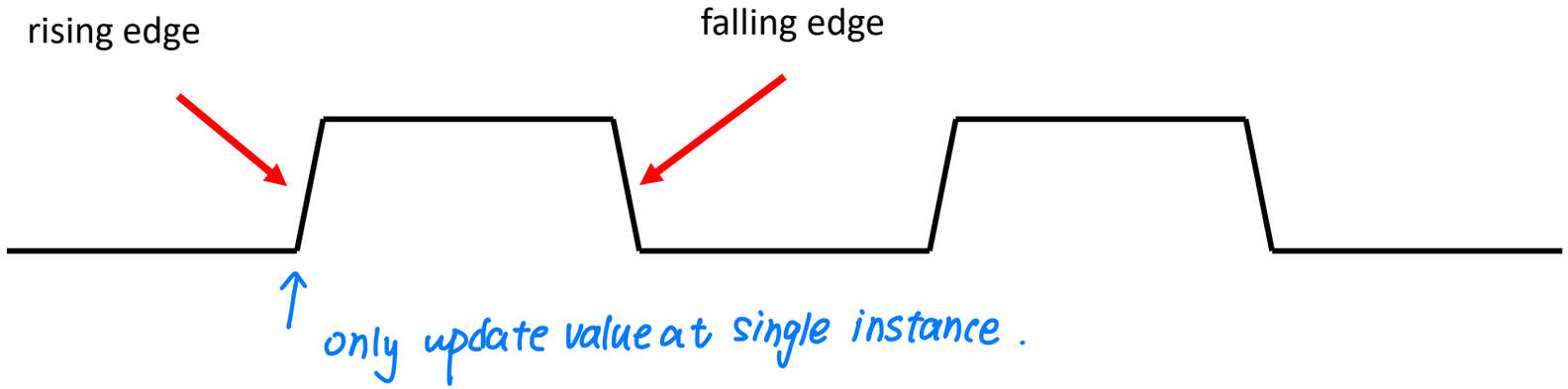
- Problem: G must be set very precisely
  - Set high for too short: latch doesn't have enough time for feedback to stabilize
  - Set high for too long: Signal may propagate round twice and increment PC by 2 (or more)
- Challenging to design circuits with exactly the right durations



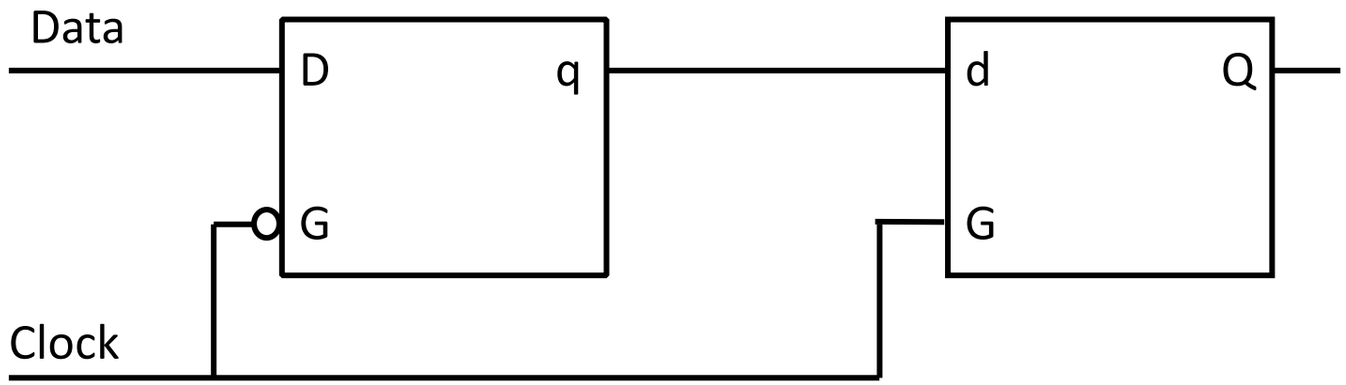
Not just a problem for PC, much of our processor will involve logic like this

Adding a Clock to the Mix

- We can solve this if we introduce a **clock**
  - Alternating signal that switches between 0 and 1 states at a fixed frequency (e.g., 1 GHz)
  - Only store the value the **instant** the clock changes (i.e. the **edge**)



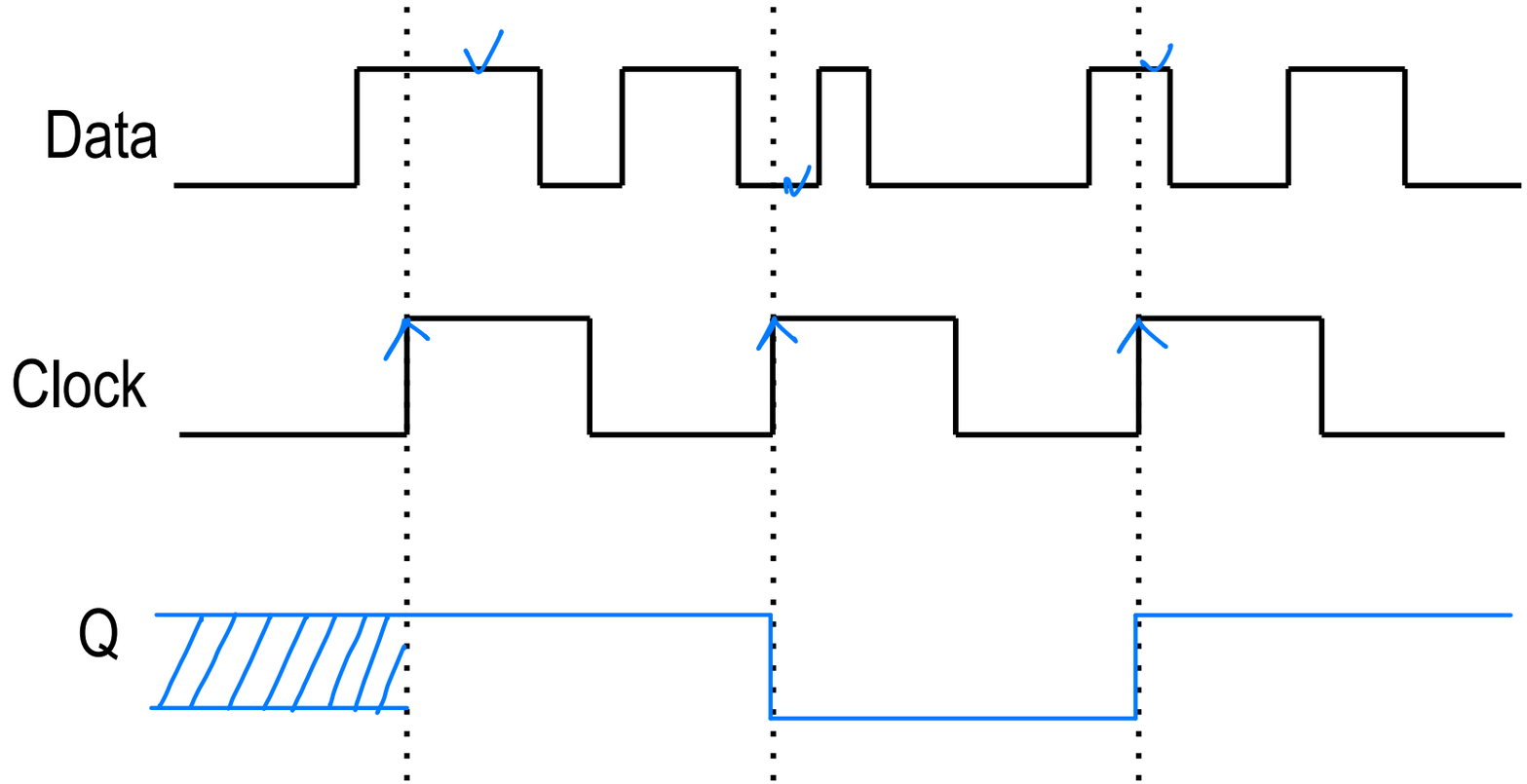
Adding a Clock to the Mix



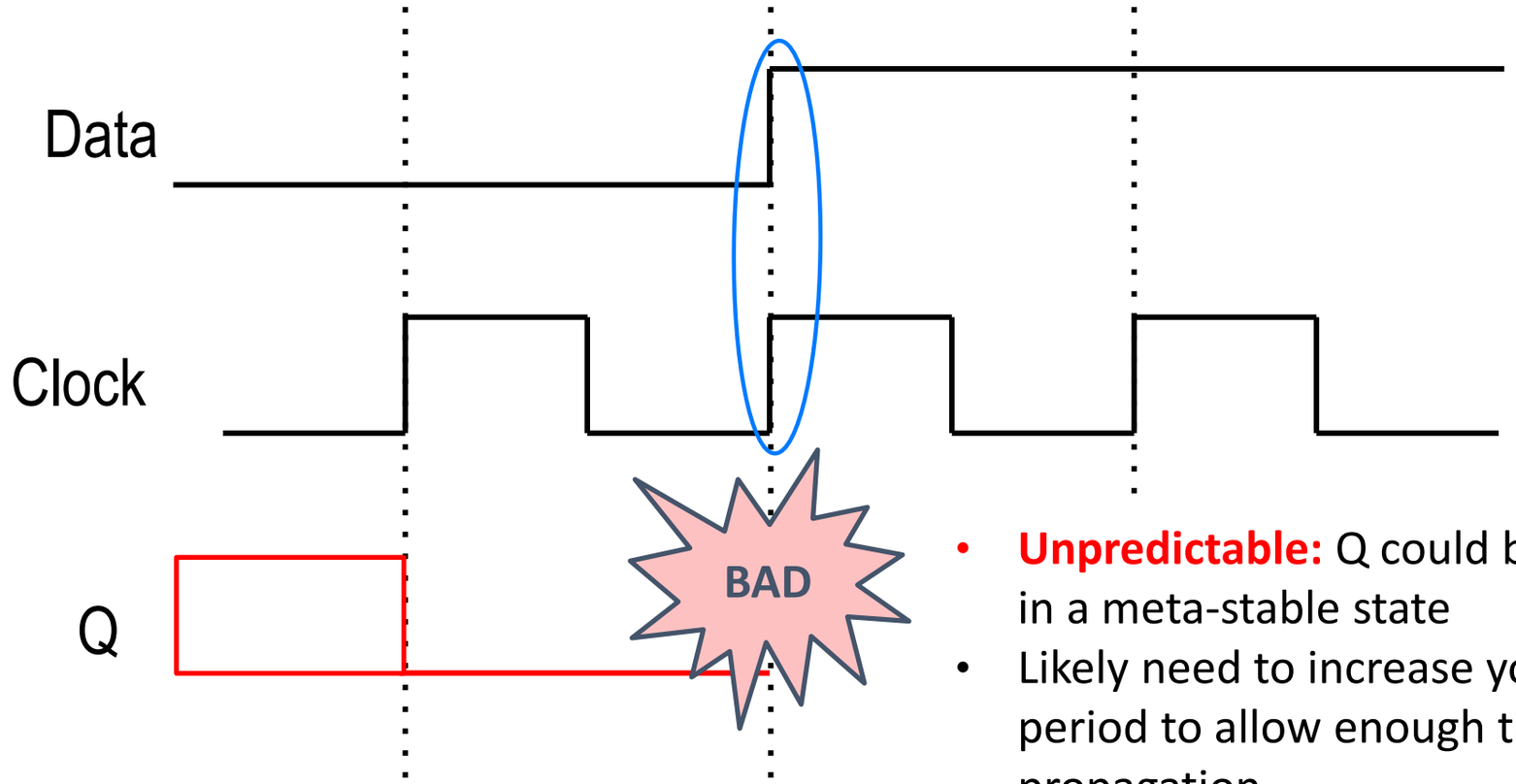
We won't discuss it further here, but this circuit sets Q=D **ONLY** when clock transitions from 0 -> 1

Intuitively, the design works by inverting the Gate signals, so only one passes at a time (like a double set of sliding doors)

D Flip-Flop Timing Diagram

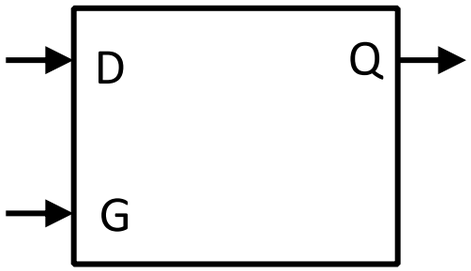


What happens if Data changes on clock edge?

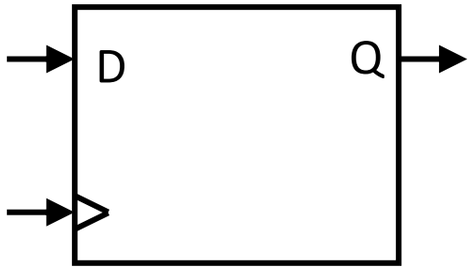


- **Unpredictable:** Q could be high, low or in a meta-stable state
- Likely need to increase your clock period to allow enough time for signal propagation

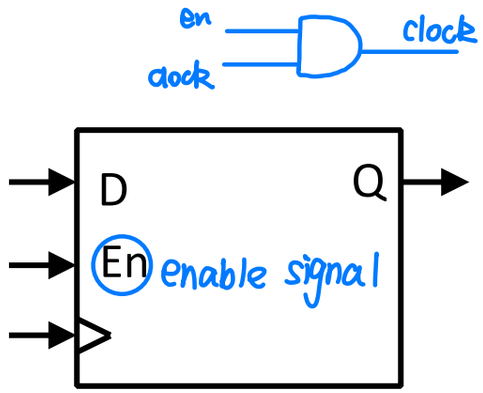
Latches vs Flip-flops



D Latch



D Flip-flop



Enabled D Flip-flop (only updates on clock edge if 'en' is high)

Finite State Machines

- So far we can do two things with gates:
  1. Combinational Logic: implement Boolean expressions
     - Adder, MUX, Decoder, logical operations etc
  2. Sequential Logic: store state
     - Latch, Flip-Flops
- How do we combine them to do something interesting?
  - Let's take a look at implementing the logic needed for a vending machine
  - Discrete states needed: remember how much money was input
    - Store sequentially
  - Transitions between states: money inserted, drink selected, etc
    - Calculate combinationally or with a control ROM *(more on this later)*

Input and Output

- Inputs:
  - Coin trigger
  - Refund button
  - 10 drink selectors
  - 10 pressure sensors
    - Detect if there are still drinks left
- Outputs:
  - 10 drink release latches
  - Coin refund latch

Operation of Machine

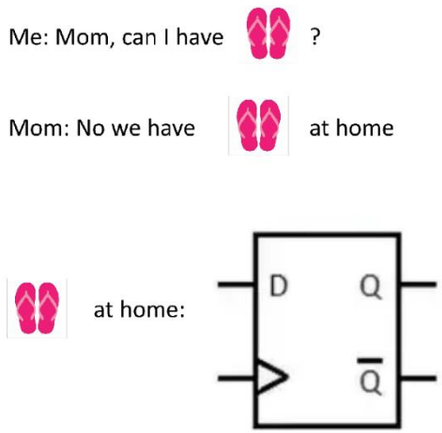
- Accepts quarters only
- All drinks are $0.75
- Once we get the money, a drink can be selected
- If they want a refund, release any coins inserted

- No free drinks!
- No stealing money.

↳ two output

Next Time

- Introduce first processor implementation



Extra Slides

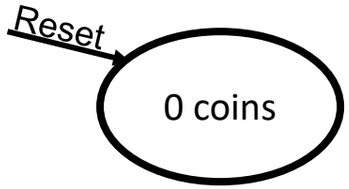
Building the controller

- Finite State Machine
  - An abstract model describing how the machine should be have under a fixed set of circumstances (i.e. finite states)
  - Remember how many coins have been put in the machine and what inputs are acceptable
- Read-Only Memory (ROM)
  - A cheaper way of implementing combinational logic
  - Define the outputs and state transitions
- Custom combinational circuits
  - Reduce the size (and therefore cost) of the controller

Finite State Machines

- A Finite State Machine (FSM) consists of:
  - K states: S = {s1, s2, … ,sk}, s1 is initial state
  - N inputs: I = {i1, i2, … ,in}
  - M outputs: O = {o1, o2, … ,om}
  - Transition function T(S,I) mapping each current state and input to next state
  - Output Function P(S) or P(S,I) specifies output
    - P(S) is a Moore Machine
    - P(S,I) is a Mealy Machine

FSM for Vending Machine



Coin trigger

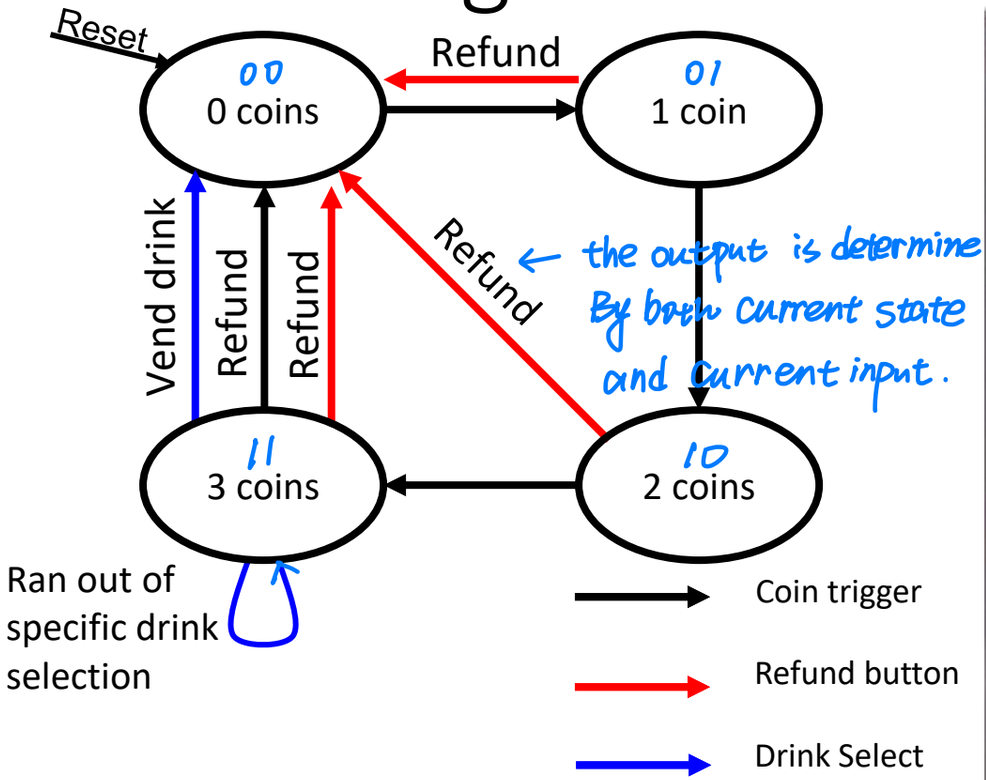
Refund button

Drink Select

FSM for Vending Machine

2

Poll: How many flip-flops would we need to remember which state we're in?



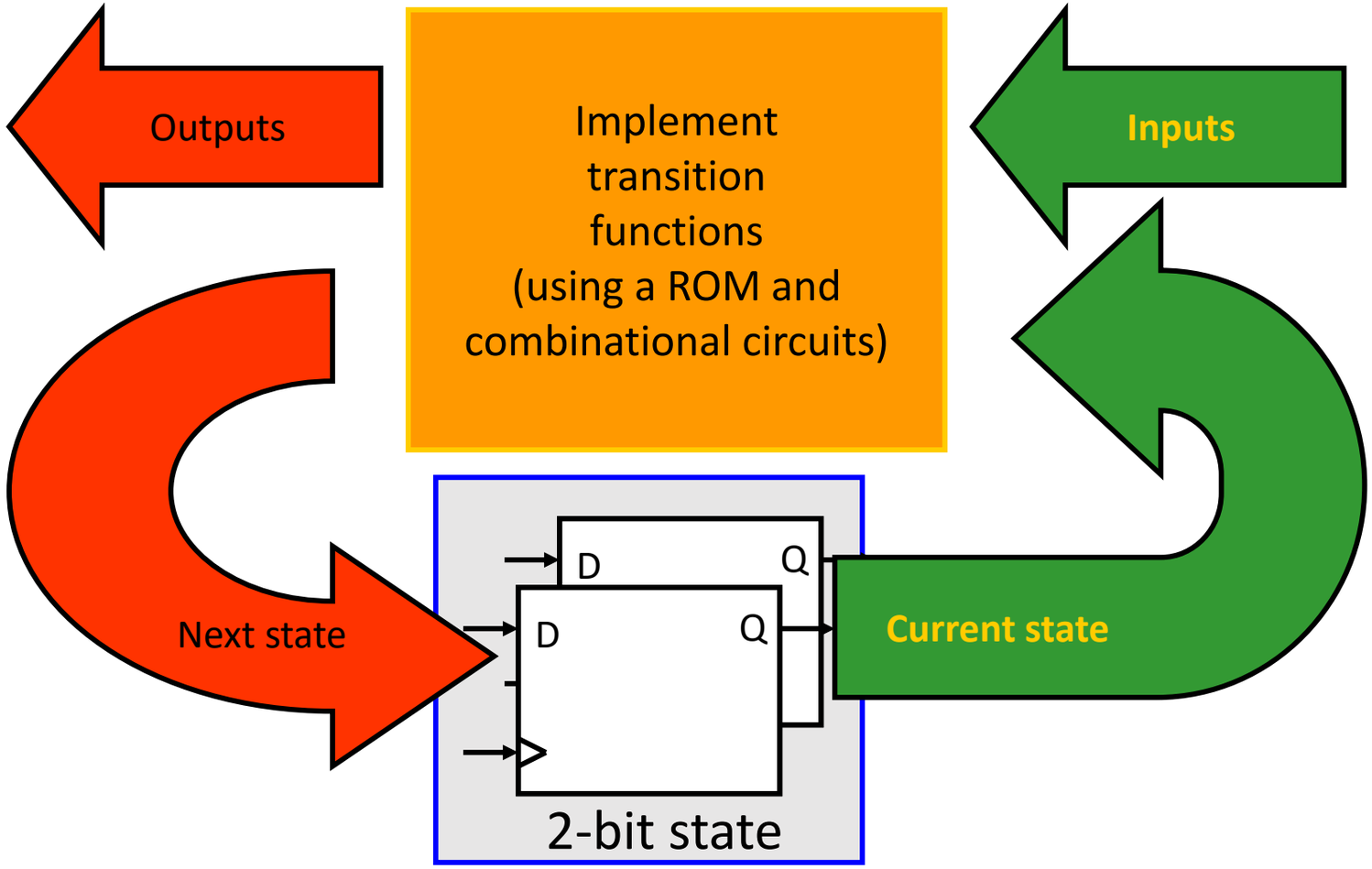
Is this a Mealy or Moore Machine?

This is Mealy: Mealy output is based on current state *AND* input

Poll: Mealy or Moore?

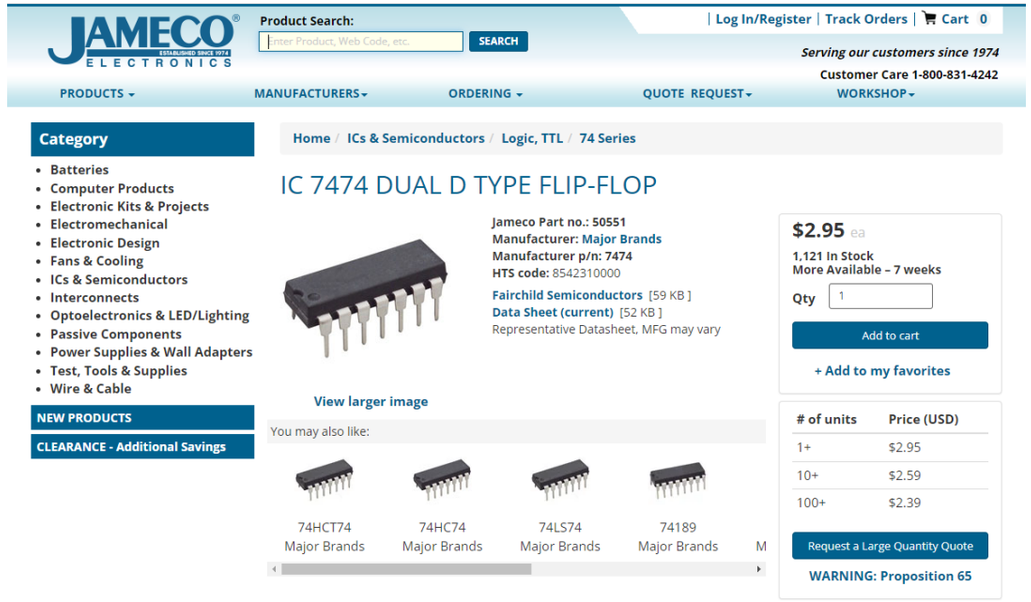
Implementing an FSM

Poll: How cheaply do you think we can build one of these controllers?



Implementing an FSM

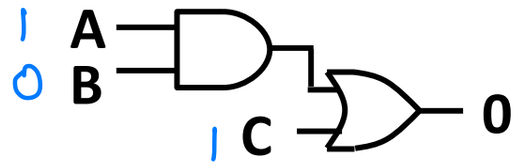
- Let's see how cheap we can build this vending machine controller!
- Jameco.com sells electronic chips we can use
  - D-Flip-flops: $3, includes several in one package
- For custom combinational circuits, would need to design and send to a fabrication facility
  - Thousands or millions of dollars!!
  - Alternative?



Implementing Combinational Logic

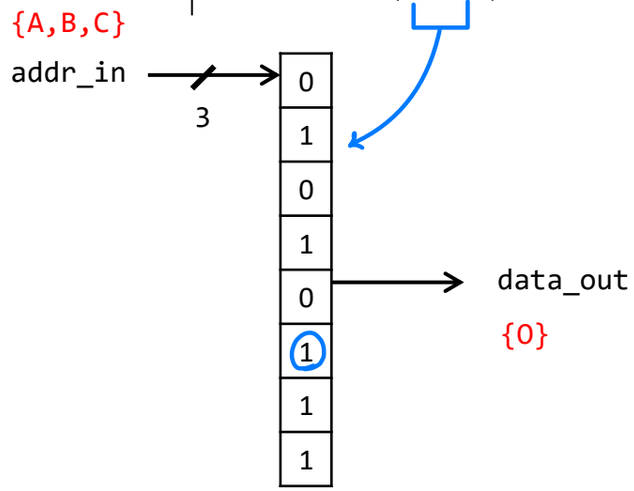
If I have a truth table:

- Either implement this using combinational logic:



101 = 5

A	B	C	O
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



- …or literally just store the entire truth table in a memory and "index" it by treating the input as a number!
  - Can be implemented cheaply using "Read Only Memories", or "ROMS"

ROMs and PROMs

IC 28C256-15 EEPROM 256K-Bit CMOS Parallel

View larger image

Jameco Part no.: 74843
Manufacturer: Major Brands
Manufacturer p/n: 28C256-15
HTS code: 8542320050

Data Sheet (current) [116 KB]
Data Sheet (current) [499 KB]
ST MICRO [62 KB]
Atmel [371 KB]
Atmel [67 KB]
Representative Datasheet, MFG may vary

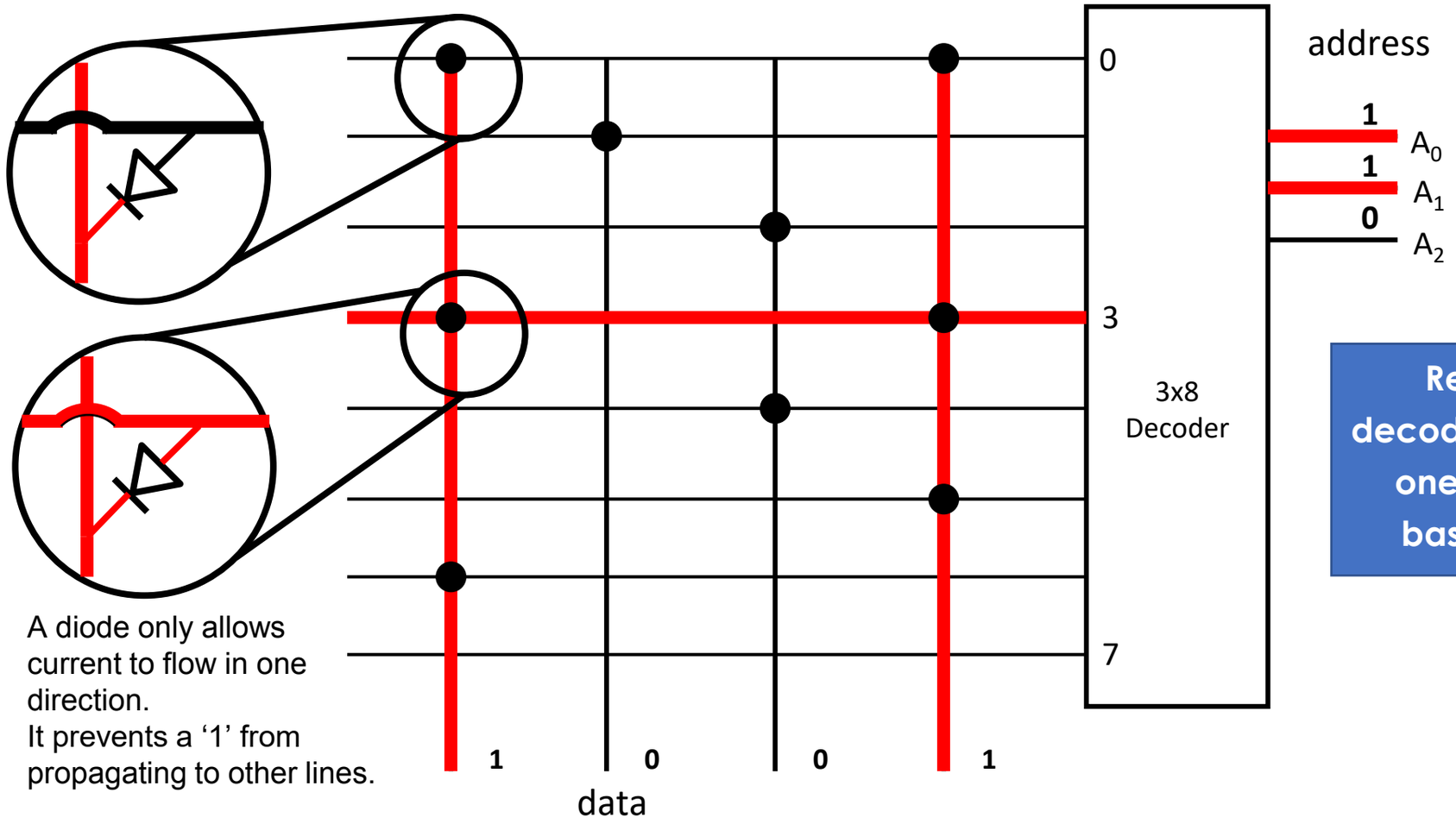
$12.25 ea
36 In Stock
More Available – 7 weeks
Qty 1
Add to cart
+ Add to my favorites

unlike a flip flop. it can store bit But we need to supply constant voltage, and if the current lost, then the bit disappears. However ROM even if lose the power, it will still hold it.

- Read Only Memory (ROM)
  - Array of memory values that are constant
  - Non-volatile (doesn't need constant power to save values)
- Programmable Read Only Memory
  - Array of memory values that can be written exactly once
- Electronically Erasable PROM (EEPROM)
  - Can write to memory, deploy in field
  - Use special hardware to reset bits if need to update

- 256 KBs of EEPROM costs ~$10 on Jameco
  - Much better then spending thousands on design costs unless we're gonna make **tons** of these

8-entry 4-bit ROM



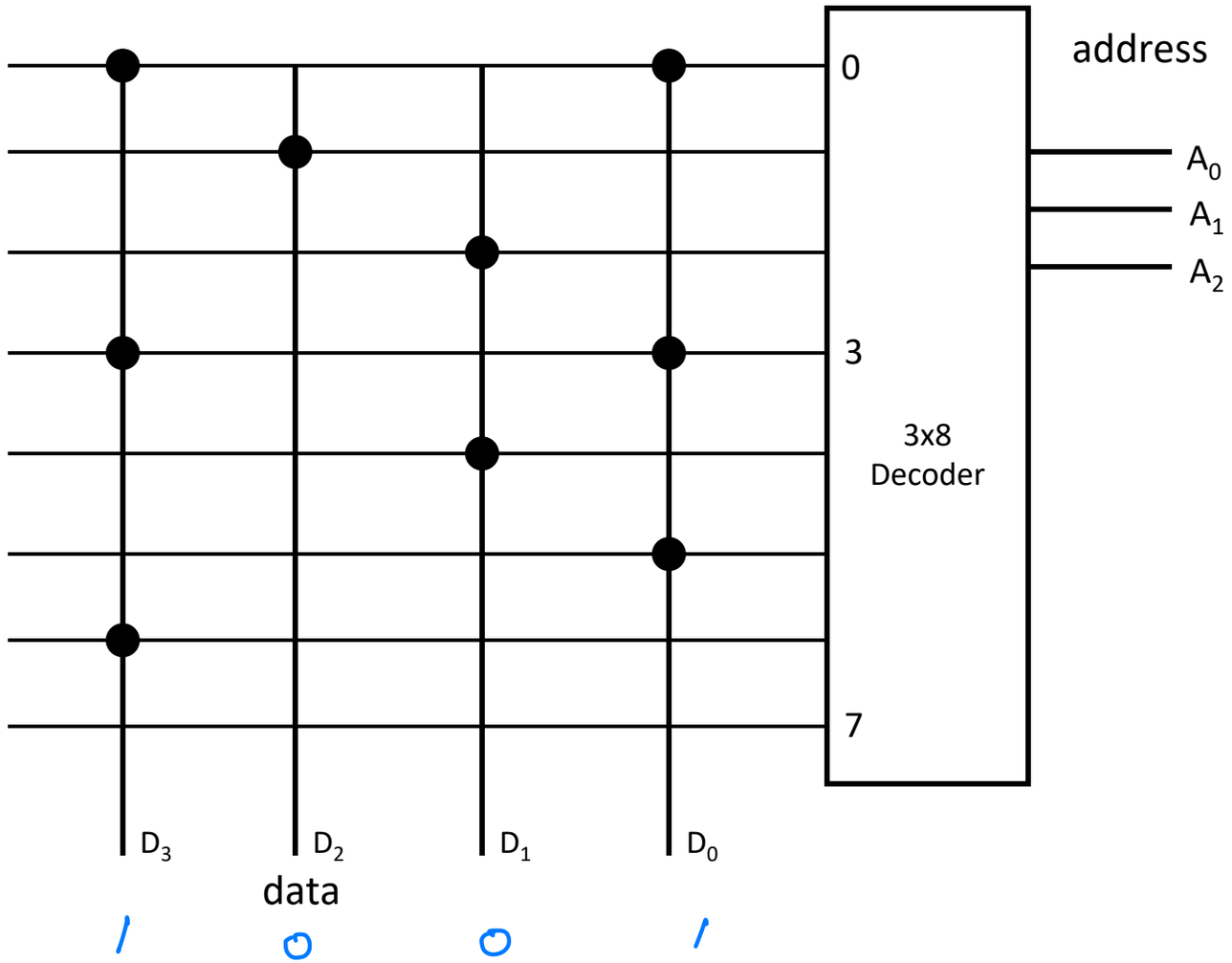
- A diode only allows current to flow in one direction.
- It prevents a ‘1’ from propagating to other lines.

Reminder: A decoder sets exactly one output high based on input

8-entry 4-bit ROM

Input	Output
000	1001
001	0100
010	0010
011	1001
100	0010
101	0001
110	1000
111	0000

This ROM corresponds to this truth table



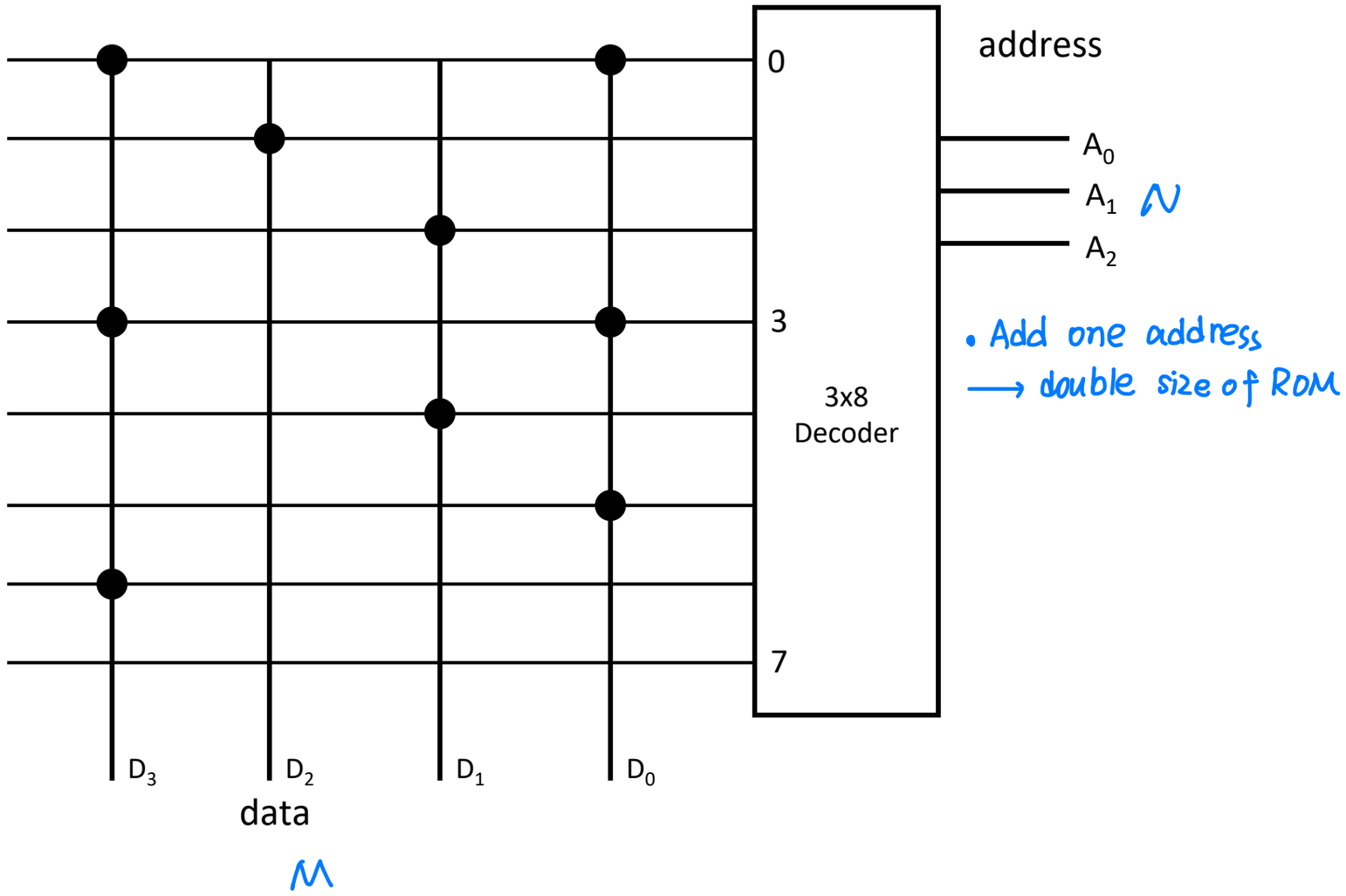
8-entry 4-bit ROM

Poll: What's the formula for size of ROM needed?

$2^N \cdot M$

Input	Output
000	1001
001	0100
010	0010
011	1001
100	0010
101	0001
110	1000
111	0000

This ROM corresponds to this truth table



- Add one address ⟶ double size of ROM

Aside: Other Memories

- Static RAM (random access memory)
  - Built from sequential circuits
    - Takes 4-6 transistors to store 1 bit
    - Fast access (< 1 ns access possible)

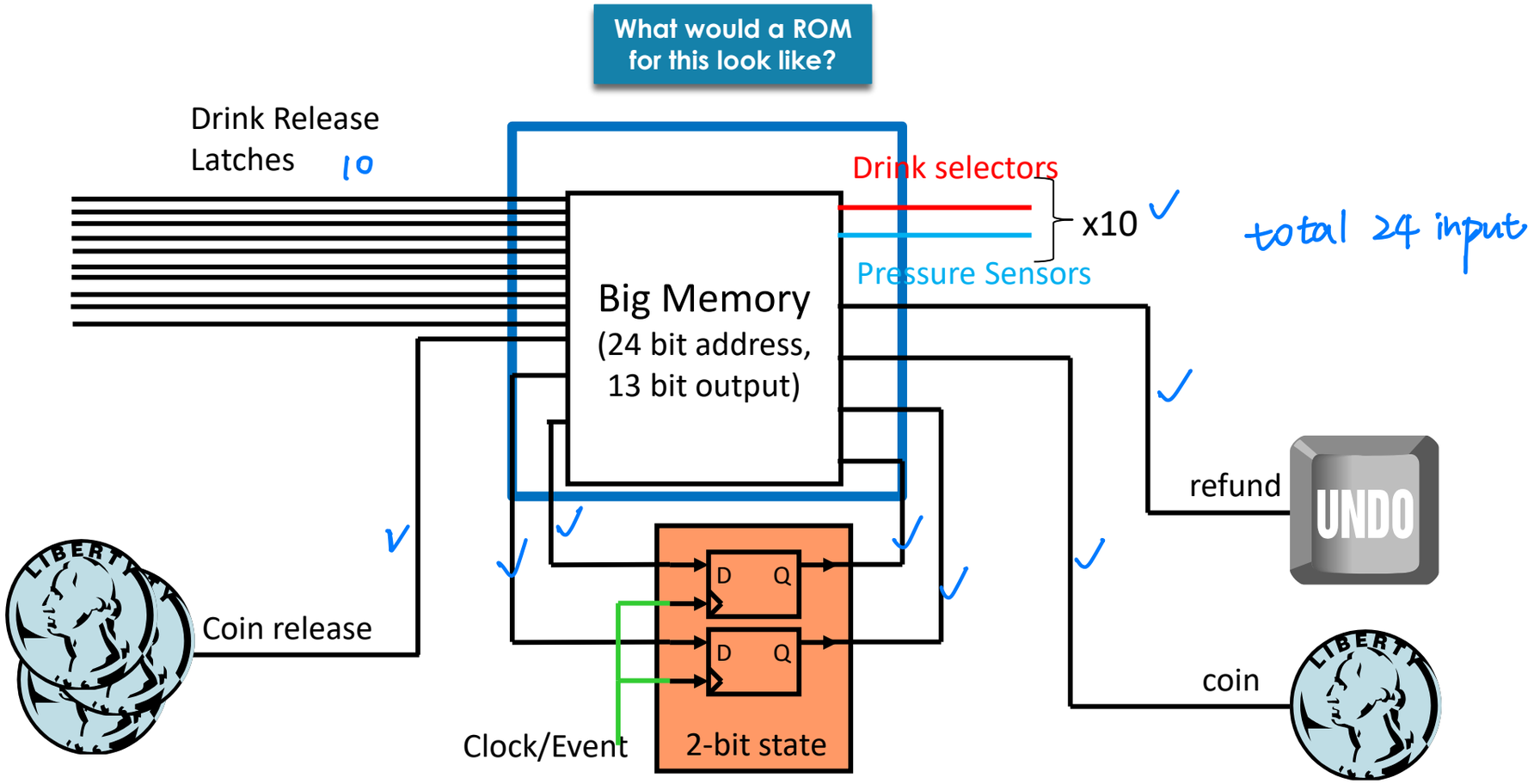
- Dynamic RAM
  - Built using a single transistor and a capacitor
    - 1's must be refreshed often to retain value
    - Slower access than static RAM
    - Much more dense layout than static RAM

- Both require constant power, or they will lose their data (i.e. are **volatile**)
- These will be used to build computer memory hierarchies (later in class)

Implementing Combinational Logic

- Custom logic
  - Pros:
    - Can optimize the number of gates used
  - Cons:
    - Can be expensive / time consuming to make custom logic circuits
- Lookup table:
  - Pros:
    - Programmable ROMs (Read-Only Memories) are very cheap and can be programmed very quickly
  - Cons:
    - Size requirement grows exponentially with number of inputs (adding one just more bit **doubles** the storage requirements!)

Controller Design So far



ROM for Vending Machine

Size of ROM is (# of ROM entries * size of each entry)

- # of ROM entries = $2^{input\_size} = 2^{24}$
- Size of each entry = output size = 13 bits

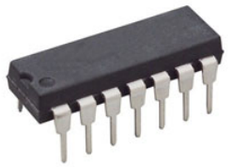
We need 2^{24} entry, 13 bit ROM memories

- **218,103,808 bits of ROM (26 MB)**
- Biggest ROM I could find on Jameco was 4 MB @ $6
  - Need 7 of these at $42??
- Let's see if we can do better

Reducing the ROM needed

- Idea: let's do a hybrid between combinational logic and a lookup table
  - Use basic hardware (AND / OR) gates where we can, and a ROM for everything more complicated
  - AND / OR gates are mass producible & cheap!
    - ~$0.15 each on Jameco

IC 74HC08 QUAD 2-INPUT POSITIVE AND GATE



Jameco Part no.: 45225
Manufacturer: Major Brands
Manufacturer p/n: 74HC08
HTS code: 8542390000

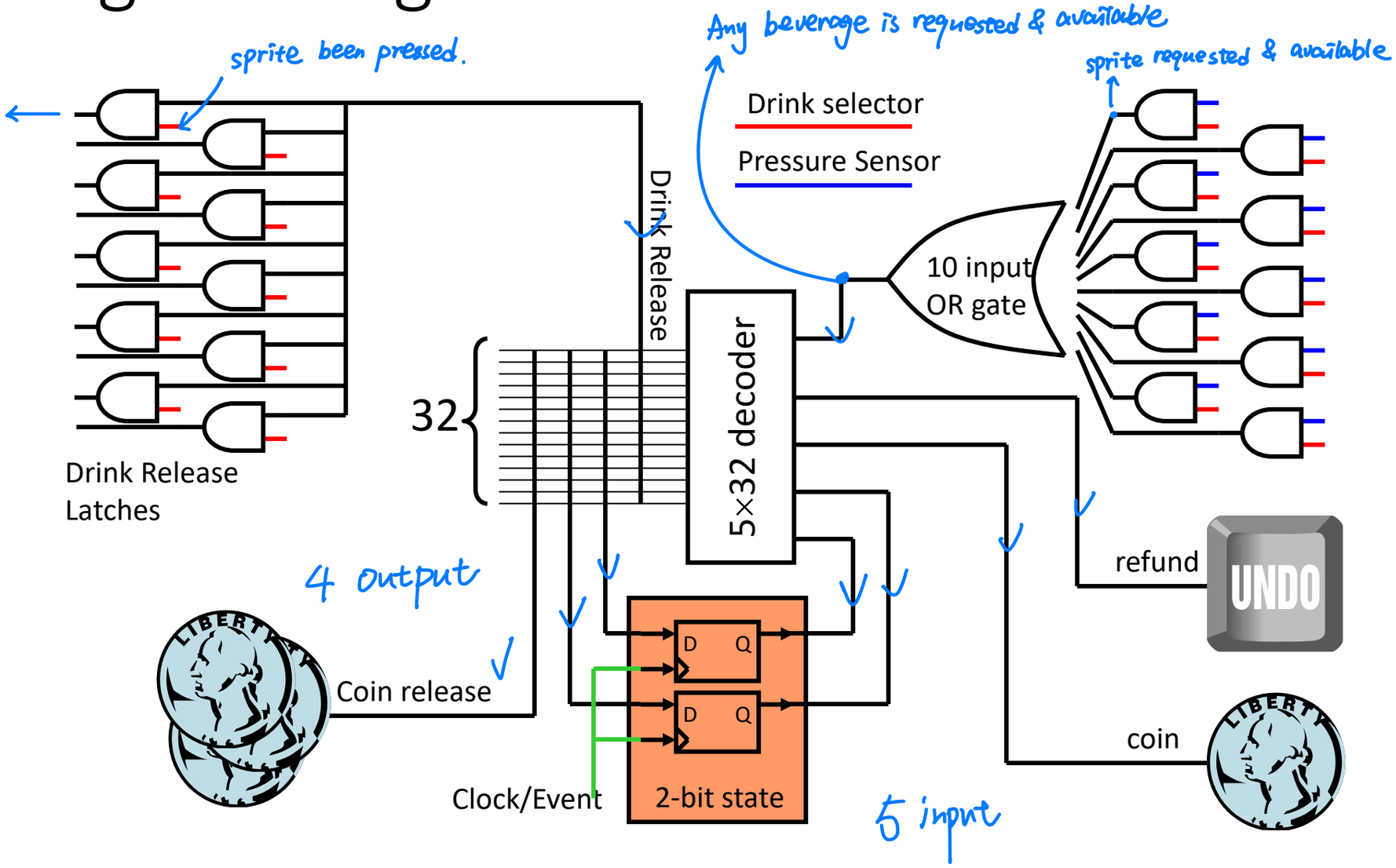
Fairchild Semiconductors [83 KB]
Data Sheet (current) [83 KB]
Representative Datasheet, MFG may vary

$0.49 ea
1,061 In Stock
More Available – 7 weeks
Qty 1
Add to cart
+ Add to my favorites

Reducing the ROM needed

- Observation: overall logic doesn't really need to distinguish between **which** button was pressed
  - That's only relevant for choosing **which** latch is released, but overall logic is the same
- Replace 10 selector inputs and 10 pressure inputs with a **single** bit input (drink selected)
  - Use drink selection input to specify which drink release latch to activate
  - Only allow trigger if pressure sensor indicates that there is a bottle in that selection. (10 2-bit ANDs)

Putting it all together



Total cost of our controller

- Now:
  - 2 current state bits + 3 input bits (5 bit ROM address)
  - 2 next state bits + 2 control trigger bits (4 bit memory)
  - $2^5 \times 4 = 128$ bit ROM
    - 1-millionth size of our 26 MB ROM 😳
- Total cost on Jameco:
  - Flip-flops to store state: $3
  - ROM to implement logic: $3
  - AND/OR gates: $5
  - **Total:** **$11**
- Could probably do a lot cheaper if we buy in bulk